Конспект практического занятия от 18.10.2024

На практическом занятии будет произведены замеры времени работы функций `sum_round_robin_aligned` `sum_round_robin` `sum_omp_reduce`.

Строка компиляции для g++ компилятора листинга 2.1 на intel будет иметь вид:
`g++ listing\ 3.cpp -o out.exe --std=c++17 -fopenmp`*

- для запуска в MinGW Installer необходимо установить `mingw32-pthreads-w32`.

Листинг 3

```
#include <iostream>
#include <cstring>
#include <vector>
#include <iomanip>
#include <omp.h>
#include <stdlib.h>
#include <new>

#define N (1u << 28)

#if defined (__GNUC__)&&__GNUC__<14
    #define hardware_destructive_interference_size 64
#else
    #include <thread>
    using std::hardware_destructive_interference_size
#endif

struct partial_sum_t {
    alignas(CACHE_LINE) unsigned val;
};

unsigned sum_round_robin_aligned(const unsigned* v, unsigned n) {
    unsigned sum = 0;
    partial_sum_t* partial_sums;
    unsigned T;

    #pragma omp parallel
    {
        unsigned t = omp_get_thread_num();

        #pragma omp single
        {
        T = omp_get_num_threads();
            partial_sums = (partial_sum_t*)calloc(sizeof partial_sums[0], T);
        }

        for (unsigned i = t; i < n; i += T)
            partial_sums[t].val += v[i];
    }

    for (unsigned i = 0; i < T; i++)
        sum += partial_sums[i].val;
```

```
    free(partial_sums);
    return sum;
}

unsigned sum_round_robin(const unsigned* v, unsigned n) {
    unsigned sum = 0;
    unsigned* partial_sums;
    unsigned T;

    #pragma omp parallel
    {
        T = omp_get_num_threads();
        unsigned t = omp_get_thread_num();

        #pragma omp single
        {
            partial_sums = (unsigned*) calloc(sizeof v[0], T);
        }

        for (unsigned i = t; i < n; i += T)
            partial_sums[t] += v[i];
    }

    for (unsigned i = 0; i < T; i++)
        sum += partial_sums[i];
    free(partial_sums);
    return sum;
}

unsigned sum_omp_reduce(const unsigned* v, unsigned n) {
    unsigned sum = 0;

    #pragma omp parallel for reduction(+ :sum)
    for (int i = 0; i < n; i++)
        sum += v[i];
    return sum;
}

unsigned sum_seq(const unsigned* v, unsigned n) {
    unsigned sum = 0;
    for (int i = 0; i < n; i++)
        sum += v[i];
    return sum;
}

int main(int argc, char** argv)  {
    auto V = std::make_unique<unsigned[]>(N);
    for (size_t i = 0; i < N; ++i)
        V[i] = i;

    auto t0 = omp_get_wtime();
    auto sum = sum_round_robin_aligned(V.get(), N);
    auto t1 = omp_get_wtime();
    std::cout << "0x" << std::hex << sum << "; " << (t1 - t0) * 1E+3 << "ms." << std::endl;

    t0 = omp_get_wtime();
    auto sum1 = sum_seq(V.get(), N);
    t1 = omp_get_wtime();
```

```
    std::cout << "0x" << std::hex << sum1 << "; " << (t1 - t0) * 1E+3 << "ms." << std::endl;

    t0 = omp_get_wtime();
    auto sum2 = sum_round_robin(V.get(), N);
    t1 = omp_get_wtime();
    std::cout << "0x" << std::hex << sum2 << "; " << (t1 - t0) * 1E+3 << "ms.";

    return 0;
}
```

Листинг 3.1 - Замер времени работы блока

```
t0 = omp_get_wtime();
...
t1 = omp_get_wtime();
std::cout << (t1 - t0) * 1E+3 << "ms.";
```

Листинг 3.2 - определение N = 2^28

```
#define N (1u << 28)
```

Листинг 3.3 - директивы

```
#if defined (__GNUC__)&&__GNUC__<14
    #define hardware_destructive_interference_size 64
#else
    #include <thread>
    using std::hardware_destructive_interference_size
#endif
```

Этот код проверяет версию компилятора GCC и, если она меньше 14, вручную задаёт значение `hardware_destructive_interference_size` равным 64. Для более новых версий GCC или других компиляторов используется стандартное определение из заголовка `<thread>`. Это обеспечивает совместимость кода с различными версиями компиляторов и доступ к информации о размере линии кэш-памяти.